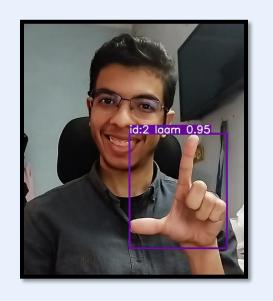
Arabic Sign Language Real-Time Recognition



(Using YOLOv8 Object Detection)

Table of contents

01

Arabic Sign Language (ASL)

04

Model Fine-Tuning

02

YOLO Architecture

05

Metrics & Results

03

Dataset

06

Code Snippets



01

Arabic Sign Language (ASL)

Arabic Sign Language



Arabic Sign Language (ASL)

A visual-gestural language used by deaf and hard of hearing communities primarily in our Arabian world. Just like spoken languages, ASL has its own grammar, syntax, and vocabulary. It involves the use of handshapes, movements to convey meaning and communicate ideas. In recent years, there has been increased attention to developing technologies that facilitate the recognition and translation of ASL gestures, aiming to enhance accessibility and communication for individuals who use sign language.



ل - Lam

Arabic Sign Language



Arabic Letters resembled by sign language

YOLO Architecture

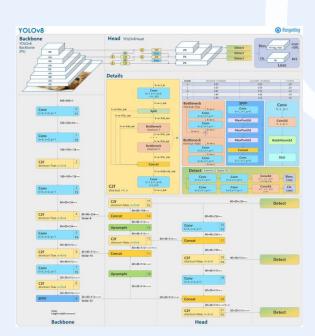


YOLO (You Only Look Once)

YOLO is known as a SOTA (State-Of-The-Art) approach and architecture that's widely used in object detection along with many other tasks. We used the YOLOv8 model

YOLO consists of many parts, mainly:

- **1. Backbone:** The backbone is the foundational part of the architecture responsible for extracting meaningful features from the input image, typically made of Convolutional Neural Networks (CNNs) such as DarkNet in YOLO.
- **2. Neck:** The neck sits between the backbone and the head. It performs additional feature fusion or transformation to enhance the representations learned by the backbone.
- **3. Head:** the head consists of convolutional layers that predict bounding boxes, objectness scores (indicating the presence of an object), and class probabilities for each grid cell in the feature map generated by the backbone and neck.



YOLOv8Architecture

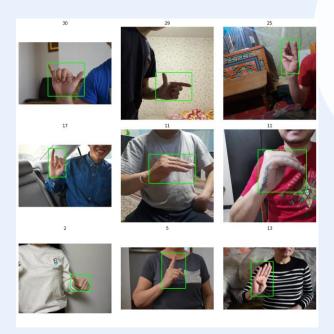
Dataset



Arabic Sign Language Dataset 2022

https://www.kaggle.com/datasets/ammarsayedtaha/arabic-sign-language-dataset-2022

The Dataset consists of 14202 images covering the 32 Arabic Signs of letters by hands made by 50 different persons, 9955 samples are used for training, while the rest 4247 samples are used for evaluating the model for generalization and performance on unseen data.



9 Plotted samples with their corresponding boundary box and label

Model Fine-Tuning



Hyper-Parameters:

Epochs: 100

Image Size: 256

batch Size: 128

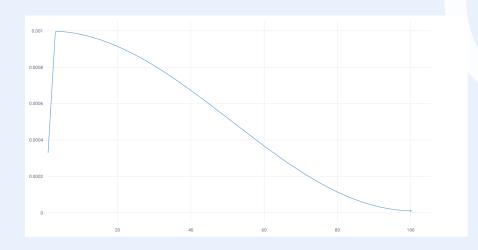
Optimizer: Adam

Initial Learning Rate: 0.001

momentum (beta for Adam): 0.9

Learning Rate Scheduler: Cosine Scheduler

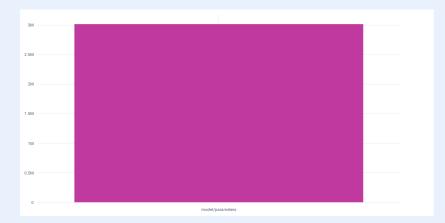
Random Seed: 42



Cosine Learning Rate Scheduler Effect

Model Fine-Tuning

Model's Parameters:





3.017 Million Parameters

8.227 **GFlOPs**



System Specs:

GPU: Nvidia RTX 3060 12GB VRAM

CPU: Intel I5-10400F 2.90 GHz

RAM: 16GB

OS: Windows 11 Pro

Frameworks:

- Visual Studio Code
- Python
- Ultralytics (YOLO)
- OpenCV







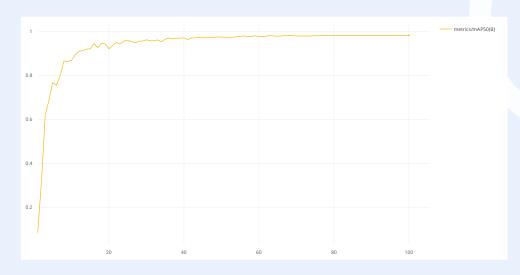






mAP50

measures the average precision of correctly identifying background regions or areas with no objects of interest in the image. It evaluates how well the model can distinguish between objects and background regions.

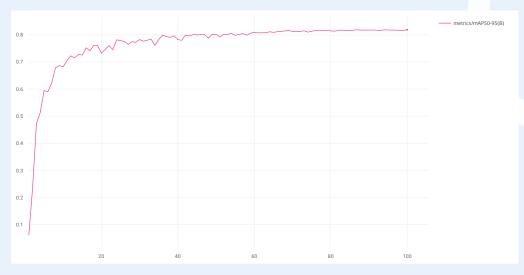


mAP50 Through the epochs 98.24% at Epoch 100



mAP50-95

calculates the average precision across a range of IoU thresholds (from 0.5 to 0.95) for background regions. It provides a broader assessment of the model's performance across various levels of overlap between predicted and ground truth bounding boxes.

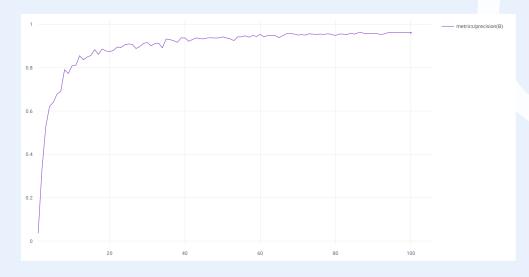


mAP50-95 Through the epochs 81.89% at Epoch 100



Precision

measures the proportion of correctly predicted background regions out of all regions classified as background by the model. It indicates the model's ability to avoid falsely identifying objects in background regions.

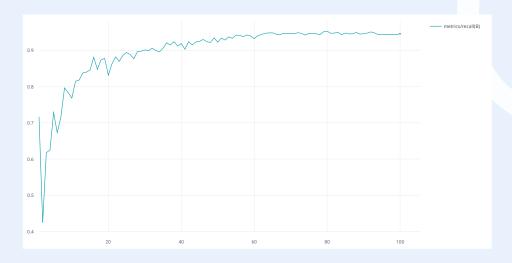


Precision Through the epochs 96.14% at Epoch 100

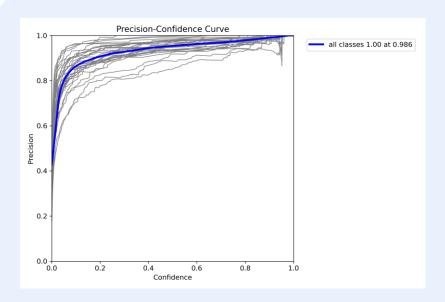


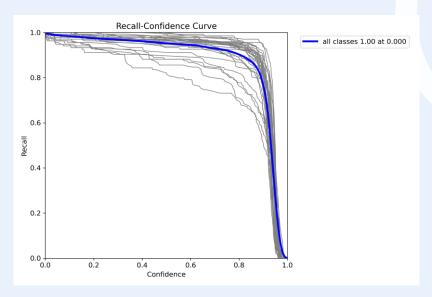
Recall

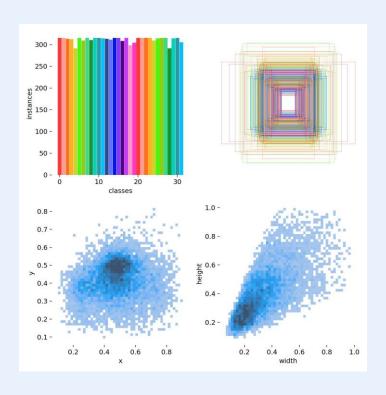
calculates the proportion of correctly identified background regions out of all actual background regions in the image. It assesses the model's ability to detect all background regions, including those that might be challenging to differentiate from objects.

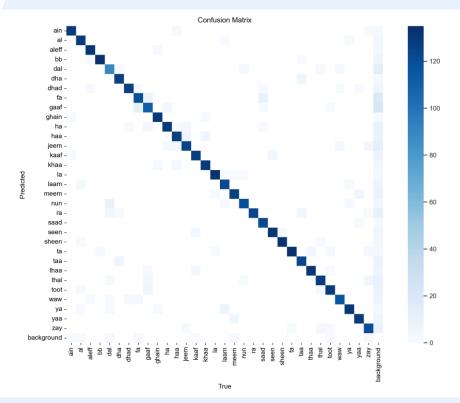


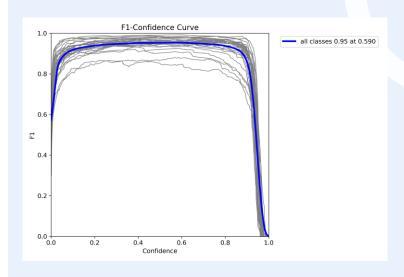
Precision Through the epochs 96.14% at Epoch 100























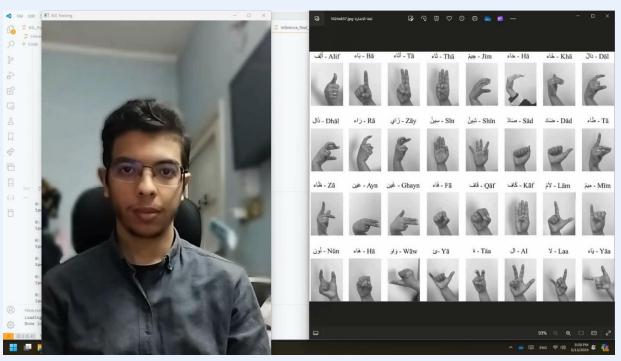








Inference of the model on some images of the Validation set



A Real-Time Inference of the model

```
from ultralytics import YOLO
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
import random
import comet_ml
comet_ml.init()
```

Imports

```
1 def read_labels(label_path):
 with open(label_path, 'r') as file:
          lines = file.readlines()
          labels = [line.strip().split() for line in lines]
7 def draw_boxes(image, labels):
8 for label in labels:
        class_id = int(label[0])
         x, y, w, h = map(float, label[1:])
       image_height, image_width, _ = image.shape
          x1 = int((x - w / 2) * image_width)
          y1 = int((y - h / 2) * image_height)
          x2 = int((x + w / 2) * image_width)
         y2 = int((y + h / 2) * image_height)
          color = (0, 255, 0) # Green
         cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)
18 return image
20 data_dir = 'train'
22 image files = [file for file in os.listdir(os.path.join(data dir, 'images')) if file.endswith('.jpg')]
24 random.shuffle(image_files)
26 fig, axes = plt.subplots(3, 3, figsize=(12, 12))
27 for ax, image_file in zip(axes.ravel(), image_files[:9]):
       image_path = os.path.join(data_dir, 'images', image_file)
       label_path = os.path.join(data_dir, 'labels', os.path.splitext(image_file)[0] + '.txt')
      # Read image
       image = cv2.imread(image_path)
       image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
      # Read LabeLs
       labels = read_labels(label_path)
38 # Draw bounding boxes on the image
       image with boxes = draw boxes(image rgb, labels)
       # Display image with bounding boxes and set the labels as titles
       ax.imshow(image_with_boxes)
       ax.set_title(str(labels[0][0]))
       ax.axis('off')
46 plt.tight_layout()
47 plt.show()
```

Visualizing Training Samples

```
1 model = YOLO('yolov8n.yaml')
2 model = YOLO('yolov8n.pt')
3 model = YOLO('yolov8n.yaml').load('yolov8n.pt')
```

```
history = model.train(data='sign.yaml', epochs=100, imgsz=256,

patience = 100, batch = 128,

project ="ASL", optimizer = 'Adam', momentum = 0.9,

cos_lr=True , seed = 42, plots = True , close_mosaic = 0, lr0 = 0.001)
```

Model Training

```
1 # Loading Model
2 trained_model = YOLO('ASL.pt')
```

```
test_images_dir = 'valid/images'
test_images = [os.path.join(test_images_dir, image) for image in os.listdir(test_images_dir)]

test_samples = np.random.choice(test_images, 10, replace=False)

print(test_samples)
```

Loading Model and Making Predictions

```
. . .
 1 results = trained_model([test_samples[0], test_samples[1], test_samples[2], test_samples[3], test_samples[4], test_samples[5], test_samples[6], test_samples[7], test_samples[8], test_samples[9]])
 3 results dir = "results tries"
 4 if not os.path.exists(results_dir):
        os.makedirs(results_dir)
7 i = 0
 8 for result in results:
        boxes = result.boxes
        masks = result.masks
        keypoints = result.keypoints
        probs = result.probs
       # Save image
        filename = os.path.join(results_dir, f"result_{i}.jpg")
        result.save(filename=filename)
        # result.show() # Uncomment to display image
        i+=1
```

Loading Model and Making Predictions

```
directory = "results_tries"

images = []

for filename in os.listdir(directory):
    if filename.endswith(".jpg"):
        img = plt.imread(os.path.join(directory, filename))
        images.append(img)

fig, axs = plt.subplots(2, 5, figsize=(50, 25))
for i in range(10):
    axs[i//5, i%5].imshow(images[i])
    axs[i//5, i%5].axis('off')

plt.show()
```

Loading Model and Making Predictions

```
. . .
1 import cv2
2 from ultralytics import YOLO
4 # Load the model
5 model = YOLO('ASL.pt')
7 # Open the camera
8 cap = cv2.VideoCapture(0)
10 # Setting width and height of the video
11 cap.set(cv2.CAP PROP FRAME WIDTH, 1920)
12 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
15 while cap.isOpened():
       success, frame = cap,read()
       if success:
          results = model.track(frame, persist=True)
          annotated_frame = results[0].plot()
          cv2.imshow("ASL Tracking", annotated_frame)
       # Break the Loop if 'q' is pressed
       if cv2.waitKey(1) & 0xFF == ord("q"):
       # Break the Loop if
35 cap.release()
```

Real-Time Inference

Thank You!